

GSM 模块 STM32 开发板使用说明

1. GSM 模块简介

GSM 是 Global System for Mobile Communications 的缩写，意为全球移动通信系统，是世界上主要的蜂窝系统之一，是现在手机所用的一种通信网络。一般 2G、3G 所指的是第二代第三代手机通信网络。在国内主要的运营商是中国移动和中国联通还有中国电信。各个运营商所是由的通信制式有所不同。移动是 GSM（2G）和 TDSCDMA（3G）；联通是 GSM（2G）和 WCDMA（3G）；电信是 cdma 2000

。以前电信还有一种叫小灵通的服务，使用的制式也是不同的。GSM 的频段分双频和 4 频，双频是 900/1800MHZ，4 频是 850/900/ 1800/ 1900 MHz.

SIM900A 模块所支持的网络是 2G 即 2 代，有些说是 2.5 代。使用频段是双频 900/1800MHZ。具体参数请查看 SIM900A 用户手册。

2. AT 指令。

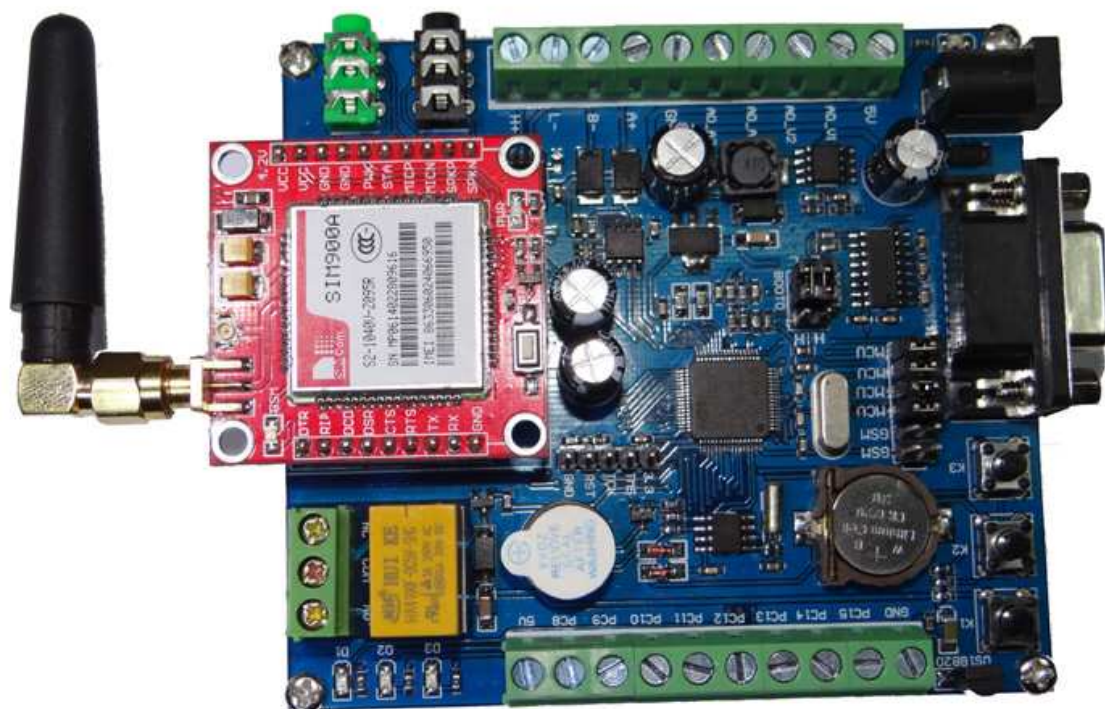
GSM 模块使用 AT 指令来操作和控制。所有向模块读写操作的命令都是以 AT 作为开始的，绝大部分是以回车结束，所组成的一条命令。AT 指令都是以字符格式发送，返回的也是字符格式。如查询模块的无线网络的信号强度是：AT+CSQ 回车，然后发送出去。

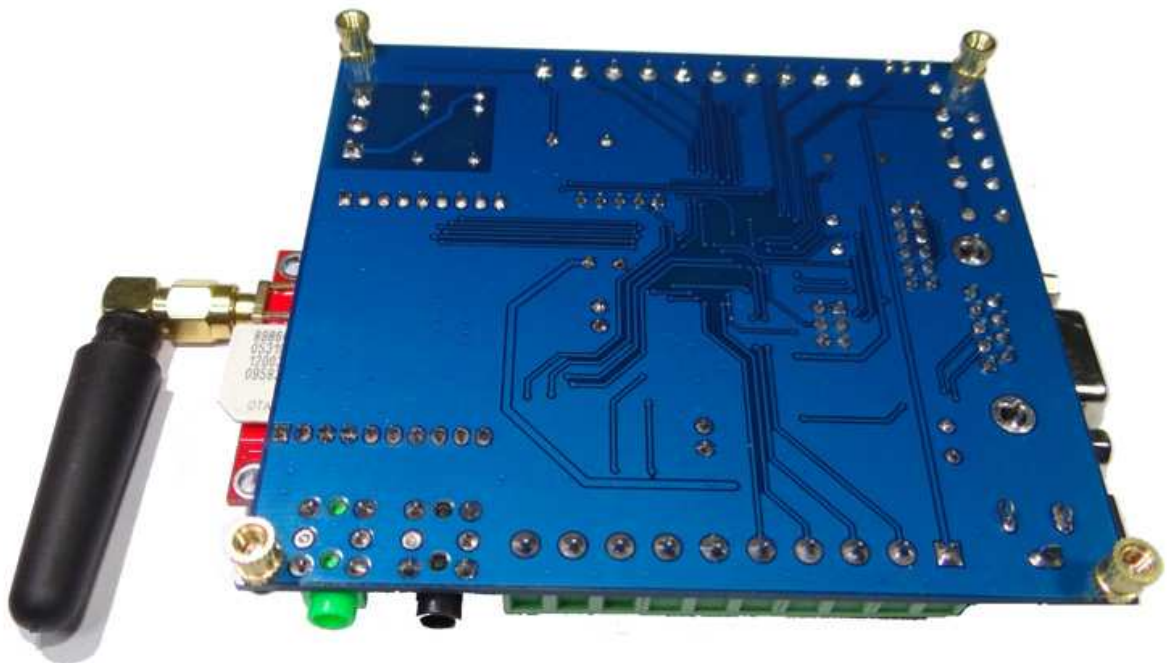
3. GSM 通信接口。

GSM 模块与外界通信都是通过串口来完成的，简单的 3 根线就能使用（TX、RX 和 GND）。其实串口有 9 个信号接口，除了上述的 3 根外，其他 6 根都是有各自的作用，而且在短信、打电话和 GPRS 链接有着十分重要的作用，能简化软件的操作，重要的是能简单准确地检测到模块的状态。

4. 开发板产品图片和功能介绍；

开发板使用现在比较流行的单片机 STM32F103RBT6 来控制 GSM 芯片。该单片机片内资源丰富，高达 72MHZ 的工作频率，128K 字节程序存储器和 20K 字节数据存储器，能满足把用户程序和转换字符的存储容量，而且丰富的接口为用户带来很多方便。单片机有 3 个串口，一个与计算机链接，另一个与 GSM 模块链接，剩下一个用于 RS485 连接。





板上的资源包括：

- 3 个用户 LED；
- 3 个用户按钮；
- 1 个继电器；
- 1 个蜂鸣器；
- 1 个串口；
- 1 个 3.5 耳机接口；
- 1 个 3.5 麦克风接口；
- 1 个 SIM 卡翻盖座；
- 1 个 RS485 接口；
- 2 个 0-10V 输入接口；
- 2 个 0-20mA 输入接口；
- 1 个 RTC 时钟芯片；
- 1 个 DS18B20 温度芯片；

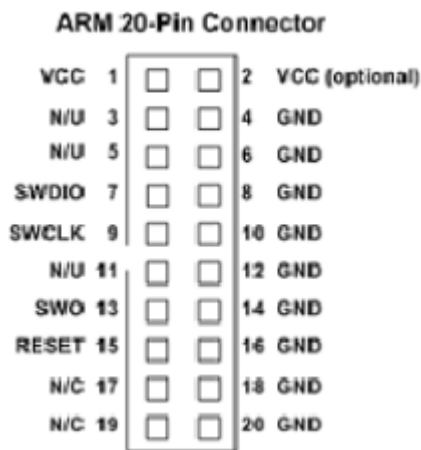
多个外接 IO。

跳线说明：跳线（MCU）是用于单片机连接 GSM 模块和单片机连接外部串口。跳线（GSM）是用于把 GSM 直接连到外部串口。

下载工具接法：

开发板只支持 SW 接口的下载方式。接上 SWDIO、SWCLK、GND 就能下载。支持 ST-LINK、ULINK、JLINK 仿真下载工具。如果是 JTAG-20PIN 的分别接 7，9，20 引脚。如果用 ULINK、JLINK 仿真，还要接上 RESET 引脚。

- **TCLK** is **SWCLK** (Serial Wire Clock)
- **TMS** is **SWDIO** (Serial Wire debug Data Input/Output)
- **TDO** is **SWO** (Serial Wire trace Output)



5. 开始学习。

对于没有单片机开发经验的，请先学习单片机开发。GSM 例程都是基于 KEIL 开发环境用 C 语言编写，调用官方的标准库 V3.5 来开发。

注意：命令都是以字符格式发送。

1. 模块的启动与初始化。参考《1-基本测试-stm32》例程。

SIM900A 第一个引脚是 PWRKEY，是硬件启动和关闭模块的功能。当模块刚上电时是处于关闭状态的，通过把 PWRKEY 引脚的电平拉低 1 秒左右，模块就会启动。模块的 52 引脚是 NET-LIGHT，即网络信号灯，模块启动后就会一闪一闪，这时就可以发 AT 指令来操作了。模块的第 66 引脚是 STATUS 即状态引脚，高电平代表模块处于开机状态，低电平处于关机状态。

当使用单片机来控制 GSM 时，发了启动信号后，要等 2 秒或网络信号灯闪烁后才能发 AT 指令，如果有数据返回，说明模块已经启动；或者通过判断‘状态引脚’来确认模块的启动。与模块通信前要先设置好波特率，模块是自动波特率的，就是当第一个向模块发送的 AT 指令的波特率作为模块使用的波特率，即模块有检测波特率的功能。例程默认是 9600 的，数据位是 8 位，1 位停止位，没有奇偶校验。

发第一条 AT 指令：AT 回车；这条指令用来同步波特率和检测 GSM 模块是否已经可以通信，当有 OK 返回，说明模块已经正常运行。

发第二条指令 AT+CSQ 回车；这是查询 GSM 无线网络的信号强度，返回的数值越高，信号越强，超过 30，信号优秀。

发第三条指令 AT+CREG? 回车；这时查询网络注册，如果有安装 SIM 卡，而且卡是支持 GSM 的，模块返回 ‘0, 1 ‘表示支持成功，返回其他的都表示是注册失败。这时一条十分重要的指令。只有确认注册成功才可以发短信、打电话和链接 GPRS 等操作。

发完 3 条指令后基本确定模块的状态，当然只发 AT+CREG? 回车这条指令确保网络注册成功也可以。现在来看看例程。

所有程序都是基于 KEIL 开发环境来编写的。程序开始先定义各个引脚，方便以后写程序。

```
/*-----*/
#include "stm32f10x.h"

/*-----*/
#define GSM_PWR_HIGH()    GPIO_SetBits(GPIOC,GPIO_Pin_5)    // 高电平-输出
#define GSM_PWR_LOW()     GPIO_ResetBits(GPIOC,GPIO_Pin_5)   // 低电平-输出

#define GSM_DTR_HIGH()    GPIO_SetBits(GPIOB,GPIO_Pin_12)    //
#define GSM_DTR_LOW()     GPIO_ResetBits(GPIOB,GPIO_Pin_12)  //

#define GSM_STATUS()      GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_4) //输入
#define GSM_RI()          GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_13) //低电平120MS是短信
#define GSM_DCD()         GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_14) //1=处于命令模式，0=数据模式

#define LED_1_ON()        GPIO_ResetBits(GPIOB,GPIO_Pin_5)    //LED低电平亮
#define LED_1_OFF()       GPIO_SetBits(GPIOB,GPIO_Pin_5)
#define LED_2_ON()        GPIO_ResetBits(GPIOB,GPIO_Pin_6)    //低电平亮
#define LED_2_OFF()       GPIO_SetBits(GPIOB,GPIO_Pin_6)
#define LED_3_ON()        GPIO_ResetBits(GPIOB,GPIO_Pin_7)    //低电平亮
#define LED_3_OFF()       GPIO_SetBits(GPIOB,GPIO_Pin_7)

#define LAY_OFF()         GPIO_ResetBits(GPIOD,GPIO_Pin_2)    //继电器
#define LAY_ON()          GPIO_SetBits(GPIOD,GPIO_Pin_2)

#define BEEP_OFF()        GPIO_ResetBits(GPIOA,GPIO_Pin_8)    //蜂鸣器
#define BEEP_ON()         GPIO_SetBits(GPIOA,GPIO_Pin_8)

#define KEY_1()           GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_0) //低电平亮
#define KEY_2()           GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_1)
#define KEY_3()           GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_2)

#define RS485_RX()        GPIO_ResetBits(GPIOA,GPIO_Pin_7)    //485
#define RS485_TX()        GPIO_SetBits(GPIOA,GPIO_Pin_7)
```

然后是在 main()函数调用第一个函数 ‘void SIM900_IO_INIT() ‘来初始化 CPU 的各个引脚，很多人认为单片机的引脚设置方向后其他默认就可以了，由于单片机的 IO 引脚很多功能复用，而且 GSM 模块和单片机使用的电压不同，存在压差，一般输入的都设成开漏输入，来适应压差，而且线路板上也有限流电阻。输出看驱动的是什么，如果是继电器、蜂鸣器等设置成上拉输出，有时也会设置成开漏输出，如单总线的驱动。


```

/*****
初始化IO,设置方向和方式
*****/
void SIM900_IO_INIT()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    //RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //开启时钟

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5; //GSM_PWR -上拉输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12; //GSM_DTR -开漏输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = (GPIO_Pin_13|GPIO_Pin_14); //GSM_RI / GSM_DCD
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //浮动输入
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4; //GSM_STATUS
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //浮动输入
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7; //RS485-DE
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //上拉输出
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

```

接着是初始化串口 IO，2 个串口一个把数据发送到计算机上显示模块初始化的信息 ‘void usart_1_init(char baud) ‘。另一个与 GSM 模块通信’ void usart_2_init(void) ‘。数据接收都是以中断方式响应。

```

/* Configure USART1 Rx as input floating */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Configure USART2 Rx as input floating */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Configure USART3 Rx as input floating */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
GPIO_Init(GPIOB, &GPIO_InitStructure);

/* Configure USART1 Tx as alternate function push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Configure USART2 Tx as alternate function push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Configure USART3 Tx as alternate function push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_Init(GPIOB, &GPIO_InitStructure);

GSM_PWR_LOW();
GSM_DTR_HIGH();
RS485_RX();

```

```
uint32_t rate;
USART_InitTypeDef USART_InitStructure;

switch(baud)
{
    case 1:rate=2400;
        break;
    case 2:rate=4800;
        break;
    case 3:rate=9600;
        break;
    case 4:rate=19200;
        break;
    case 5:rate=38400;
        break;
    case 6:rate=57600;
        break;
    case 7:rate=115200;
        break;
    default:rate=9600;
        break;
}
USART_InitStructure.USART_BaudRate = rate;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

/* Configure USART1 */
USART_Init(USART1, &USART_InitStructure);
/* Enable USART1 Receive and Transmit interrupts */
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
/* Enable the USART1 */
USART_Cmd(USART1, ENABLE);
```

现在看 MAIN 函数，见下图，主程序的开始，各个功能初始化过程，第一个是初始化系统时钟函数，由于 STM32 系列的单片机 CPU 时钟是可设置的，一般都是以最高速度运行即 72MHZ；接着是延时函数，这个不是必须的；NVIC_Configuration()这个函数是设置中断优先级，这个跟中断函数是紧密相连的；各个 IO 初始化函数，设置方向和驱动方式；串口初始化函数默认是 9600 波特率；LED 闪烁函数是告诉用户程序已经运行起来，接下来就是 GSM 模块的初始化了。

```
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
/*****
*主函数
*****/
int main(void)
{
    // unsigned char system_counter=0;
    // unsigned char i=0;
    /* System Clocks Configuration */
    RCC_Configuration(); //初始化CPU时钟
    GSM_delay_1ms(1000); //等待电源稳定
    /* NVIC configuration */
    NVIC_Configuration(); //定义各种中断优先级
    /* Configure the GPIO ports */
    SIM900_IO_INIT(); //控制GSM模块的IO初始化
    GPIO_Configuration(); //板上IO初始化
    /*****串口初始化*****/
    sim900_uart();
    /*****板上LED闪烁*****/
    led_flash(); //板上LED闪烁
    USART1_send(0);
    USART2_send(0);
    SIM900_POWER_ON(); //启动GSM模块
    syn_Baud_rate(); //同步波特率
    USART1_string_send("\nGMS-STM32 development board.\r\n");
    SIM900A_INIT(); //GSM模块初始化
    sys_Tick(); //系统滴答器初始化
    /*****
    while(1)
    {
```

GSM 模块启动函数。GSM 模块的状态引脚的读取，在头文件 ‘SIM900_IO.H’ 里已经作了声明。为什么先读取状态，如果该引脚是高电平，表示模块已经启动，再根据状态来启动模块，这是因为 STM32 下载程序的过程不需要断电，那么模块很可能已经处于启动状态，如果再向 GSM 模块的启动引脚输入低电平脉冲，模块就会关闭。即模块处于关闭状态时向启动引脚输入低电平脉冲，模块启动。既然是输入低电平，为什么程序里是高电平的？这是因为模块的启动引脚是由 NPN 三极管控制的，这个用户看一下原理图就明白。那为启动模块后要延时这么长时间？这是因为模块启动后需要一段时间来搜寻网络。

```

250  /*****
251  模块启动，读取GSM_STATUS电平，高电平=已启动，低电平=关闭
252  *****/
253  void SIM900_POWER_ON()
254  {
255      unsigned char i=0;
256      if(GSM_STATUS()==0)    //0没有启动
257      {
258          GSM_PWR_HIGH();
259          GSM_delay_1ms(1000);
260          GSM_PWR_LOW();
261          for(i=0;i<5;i++)
262          {
263              GSM_delay_1ms(1000);
264              if(GSM_STATUS()==1) //GSM_STATUS=1已启动，=0没有启动
265              {
266                  break;
267              }
268          }
269          for(i=0;i<8;i++) //等待GSM加入网络
270              GSM_delay_1ms(1000);
271      }
272      else
273      {
274          GSM_PWR_LOW();
275      }
276  }
277  }
    
```

模块启动了，先条调用 ‘syn_Baud_rate()’；‘这个函数来同步波特率，当然只要发以 AT 开头的命令都可以同步波特率；接着是 GSM 模块的初始化了’ SIM900A_INIT() ‘，这个主要是查看信号强度和网络注册情况，并通过串口输出到上位机。

现在来看看信号强度读取函数见下图，向 GSM 模块发送 ‘AT+CSQ 回车’ 指令读取信号强度，等收到返回数据后使用字符检索函数 ‘strstr(RxBuf_2,"CSQ:");’，读出想要的的数据，这个字符检索函数是在 “string.h” 里声明的，调用程序需要包含这个头文件，当要检索的数据存在的，会返回这个数据所指的开始位置，通过偏移取得想要的的数据，即信号强度数据，然后输出。信号数据越大信号越好。30 以上是很好，20 以上是一般好。为什么会重复几次？这是因为当向模块发命令时，模块还在搜寻网络，没有稳定下来，没有获取到信号强度。可不可以只查一次，这个按个人喜好吧。

```

AT+CSQ

+CSQ: 11,0
    
```

AT 命令查询和返回结果


```

57  /*****
58  *读取信号强度。
59  *****/
60  void sim900_at_csq()
61  {
62      char *buf_ok;
63      char i;
64      for(i=0;i<3;i++)
65      {
66          claer_u2_buf();
67          USART2_string_send("AT+CSQ\r");
68          check_rx2_status(11,20);
69          buf_ok=strstr(RxBuf_2,"CSQ:");
70          if(buf_ok)
71          {
72              buf_ok +=5;
73              if((*buf_ok>=0x30)&&(*buf_ok<=0x39))
74                  SIM900.dbm[0]=*buf_ok;
75              else
76                  SIM900.dbm[0]='0';
77              buf_ok++;
78              if((*buf_ok>=0x30)&&(*buf_ok<=0x39))
79                  SIM900.dbm[1]=*buf_ok;
80              else
81                  SIM900.dbm[1]=0;
82              SIM900.dbm[2]=0;
83              break;
84          }
85          GSM_delay_1ms(500);
86      }
87      claer_u2_buf();
88  }

```

GSM 信号强度读取函数

GSM 网络注册状态获取过程跟信号强度查询是一样的，这里不在陈述；

```

89  /*****
90  *检查GSM网络登记
91  *****/
92  void sim900_at_creg()
93  {
94      char count=0;
95      char *buf_ok;
96      for(count=0;count<3;count++)
97      {
98          claer_u2_buf();
99          USART2_string_send("AT+CREG?\r");
100         check_rx2_status(10,20);
101         buf_ok=strstr(RxBuf_2,"CREG: 0,1");
102         if(buf_ok)
103         {
104             SIM900.reg=1;
105             break;
106         }
107         else
108             SIM900.reg=0;
109         GSM_delay_1ms(1000); //1s
110     }
111     claer_u2_buf();
112 }

```

AT+CREG?

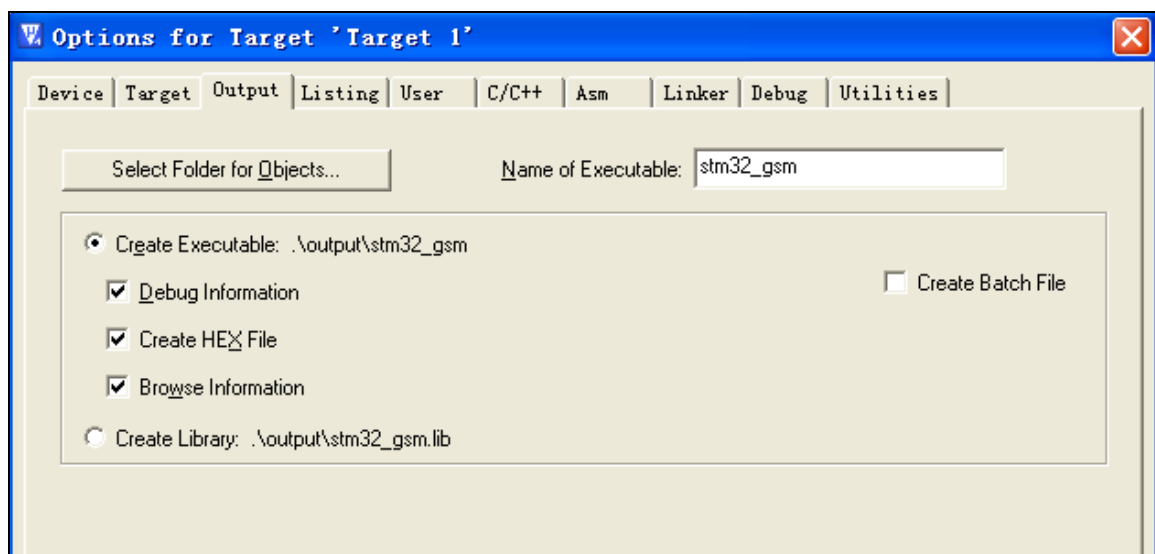
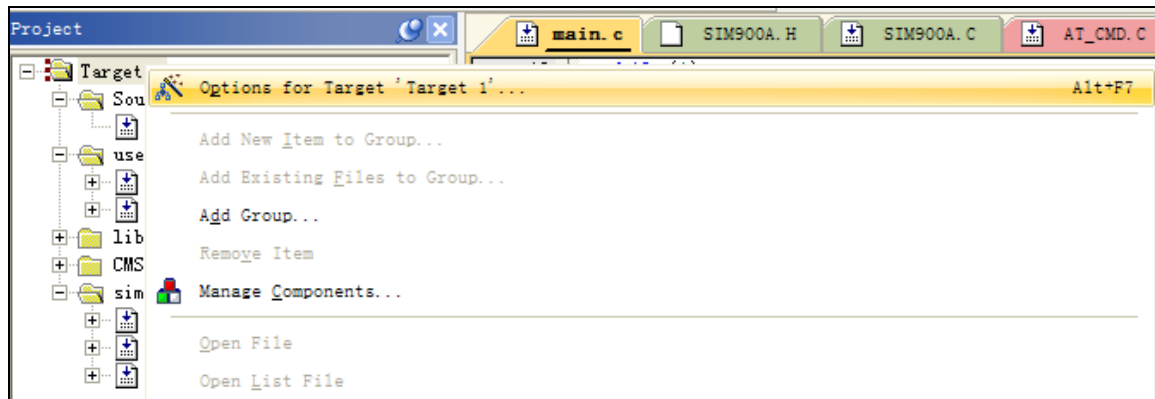
+CREG: 0,1

网络注册查询命令

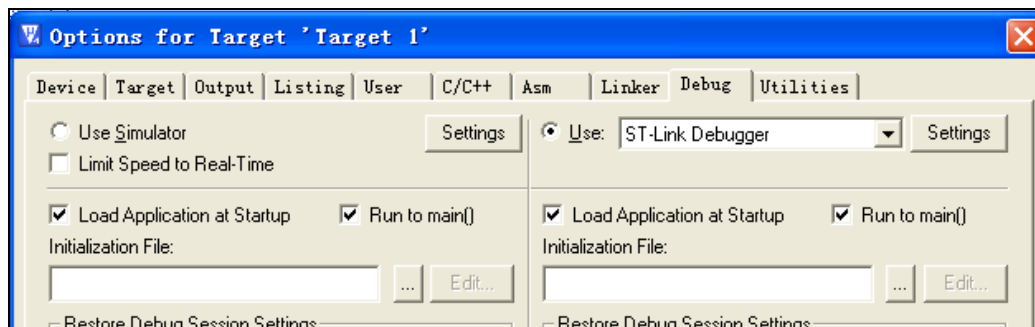
实际上当用户编写程序时根本不知道 AT 指令返回的数据是怎样解析,这个需要先看 AT 指令手册和应用文档,或者把开发板上的跳线帽跳到 GSM 处,其跳线帽他拿掉,按照命令手册,用串口调试助手,发几条常用的命令来熟悉一下。

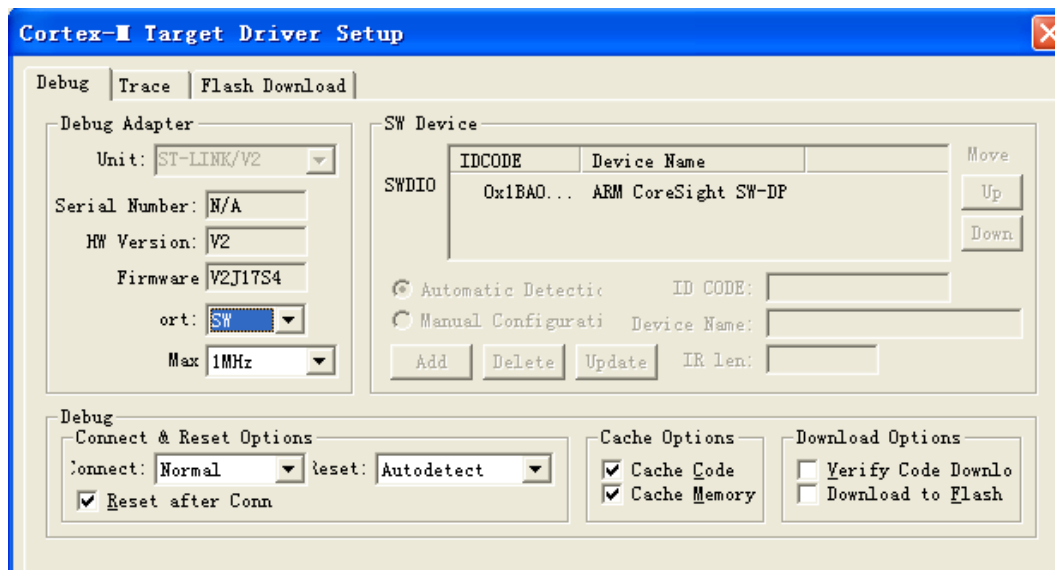
GSM 模块初始化后,即进入循环处理程序。这里只是 3 个按键响应 3 个不同的功能,是用于测试板上的继电器、蜂鸣器和 LED 灯的好坏。

一个基本的程序已经完成,编译通过后,接着是把程序下载到单片机里。先设置 KEIL 编译器,可以让编译时生产 HEX 文件。按下图选择 options for target 选项,像下图一样,勾上,在 Name of Executable 输入生产 HEX 文件的名字,不输入回事默认跟工程文件名称一样。

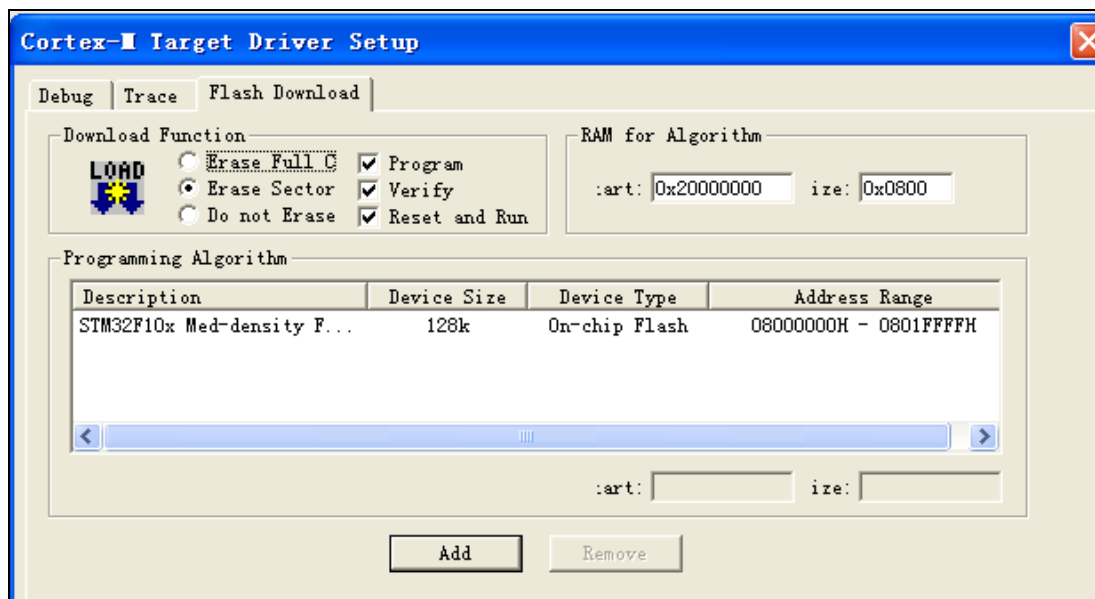
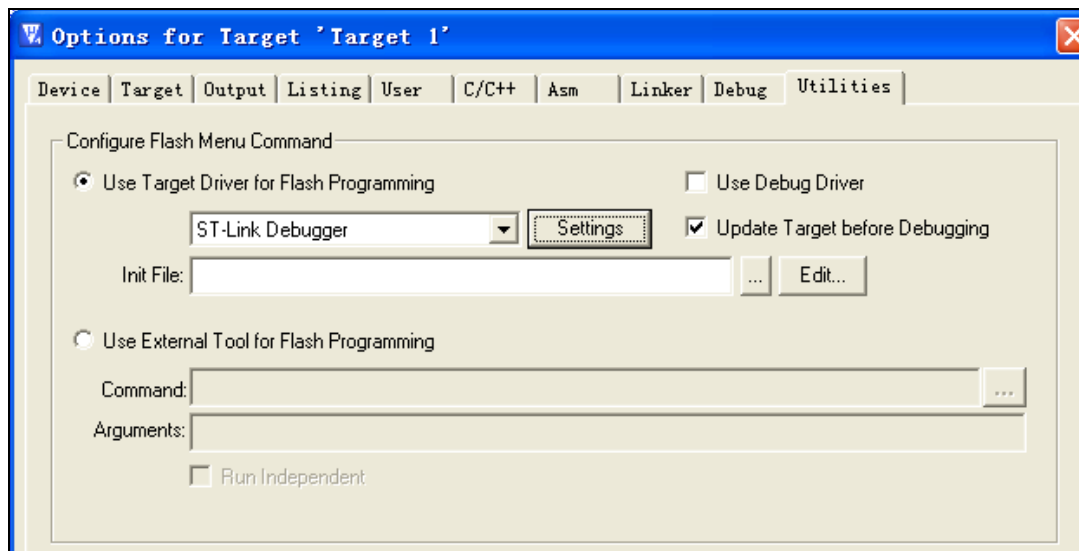




然后选择 debug, 选择使用仿真工具, 通过下拉列表框选择对应的工具, 例如 ST-LINK。然后点击 Setting 按钮, 在 port:处选择 ‘SW’, 如果硬件接线没错, 而且线路板已上电和 ST-LINK 已接上电脑, 在 SW Device 框里有 ID 号和设备名称, 否则提示没有设备。

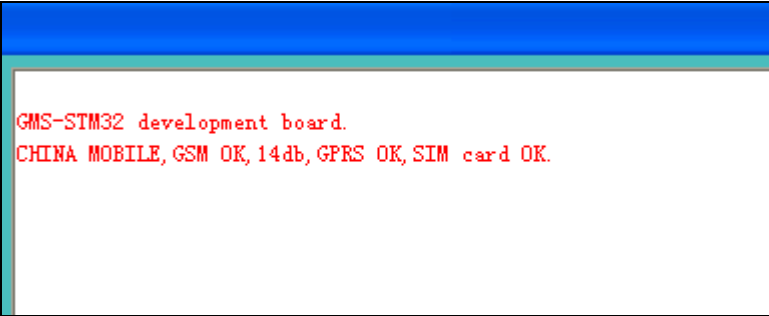




最后选择 Utilites 进入程序下载设置，这个跟前边一样，选择对应的工具，点击 Setting 按钮可以选择 FLASH 的擦除方式和单片机容量。也可以设置下载完是否运行程序。



设置完毕后，回到程序编程界面，可以点击‘LOAD’按钮也可以点击‘debug’按钮都可以把程序烧录进单片机。单片机运行，然后输出模块信息。



2. 拨打电话。参考《2-电话功能-stm32》

打电话比较简单，ATD+号码分号+回车；**注意带分号的**，命令正确返回 OK，根据不同的接通状态返回相应的信息，如忙音，没人接听或接通等。程序首先设置来电显示，有电话打进时，GSM 模块会输出号码，然后按键也相应设置成打电话（KEY_2），接电话（KEY_3）和挂断电话（KEY_1）这 3 个功能。

```
50 sys_Tick();
51 sim900a_call_init(); //设置显示来电号码
52 /*****/
53 while(1)
54 {
```

10.4 呼叫控制命令		
示范	语句	反馈结果
建议一个语音呼叫	ATD6241xxxx;	OK 移动台拨打电话

2.2.3 ATD 移动台呼叫某一号码

ATD 移动台呼叫某一号码	
执行命令 ATD<n>[<mgsml>]	<p>响应</p> <p>执行命令用于建立语音、数据或传真的主叫，还可以用于控制补充业务。说明：若在执行过程中收到 ATH 命令，则该命令可能被终止执行。但是，在建立连接的某些状态下（比如在握手状态时），该命令不会被终止执行。</p> <p>若没有拨号音并且（参数设置 ATX2 或者 ATX4） NO DIALTONE</p> <p>若占线并且（参数设置 ATX3 或者 ATX4） BUSY</p> <p>若无法建立连接 NO CARRIER</p> <p>若对方无应答 NO ANSWER</p> <p>若连接成功，且为非语音呼叫 CONNECT<text> TA 切换至数据模式</p>
	<div>ATH</div> <div>OK[挂断电话]</div>

当有电话打入时，模块返回‘RING’信息；可以选择接电话或挂断；只要对方没有挂断，就一直有‘RING’信息。而且 GSM 模块的 RI 引脚会由高电平变成低电平，直到接通或挂断才会回复高电平。就是说这个 RI 低电平的时间会很长。

被叫呼叫的示例	先与模块建立一个被叫呼叫。	RING
	ATA	RING
	ATH	OK[接听电话]
		OK[挂断电话]

当打电话时，要先设置显示主叫状态，命令是‘AT+MORING=1 回车’，模块默认该功能是关闭的。输入打电话命令，如果电话号码长度不对模块不会返回错误，也就是模块不对号码检测。有一点要注意，模块主叫，对方没有接而是挂断，模块不会也跟着挂断而是还处于呼叫状态，要等上较长一段时间才会挂断。


```
97  /******
98  打电话
99  *****/
100 char cell_num[]="10086"; //
101
102 void sim900a_call_out(void)
103 {
104     unsigned char y=0;
105     if(SIM900.call==0)
106     {
107         sim900_at_moring(); //设置显示主叫状态
108         y=sim900_at_call(cell_num); //拨号
109         if(y==1)
110             SIM900.call=1;
111     }
112 }
```

3. 发英文短信，参考《3-短信收发-stm32》例程。

在《SIM900A_AT 命令手册》里第 13 章 AT 命令演示内包含有短信发送例子，如下图，

13.7 短信命令		
示范	语句	反馈结果
设置短信系统进入文本模式，与之对应的是 PDU 模式。	AT+CMGF=1	OK
发送一条短信给自己	AT+CSCS="GSM" AT+CMGS="+861391818xxxx" >This is a test <Ctrl+Z>	OK +CMGS:34 OK

设置命令 AT+CMGF=[<mode>]	响应 设置指定短消息的输入和发送格式。 OK
	参数说明 <mode> 0 PDU 模式 1 文本模式
参考 GSM 07.05	说明

开发板上的按键 1 是发英文短信，按键 2 是发已转换编码短信，按键 3 是发动态内容的短信。

下图是发英文短信的过程，先设置为文本格式，然后设置 TE 字符集，然后写入目标号码，可以不带“+86”，注意：号码两边是引号，英文的引号，当返回是‘>’时，输入短信内容，注意不要以回车结束，然后输入‘Ctrl+Z’即十六进制的 0x1A，发送完毕返回+CMGS。例程是当按下‘按键 1’就发送一条短信，想改成自己的号码修改目标号码对应的数组内容。

```

180 /*****
181 发英文短信
182 *****/
183 char Target_number[]="13812345678"; //目标号码
184 char Target_msg[]="This is a GSM message text SMS."; //短信内容
185
186 void send_sms_text(void)
187 {
188     unsigned char b_ok=0;
189     b_ok=send_txt_sms(Target_number,Target_msg); //发送英文短信。
190     if(b_ok==0)
191     {
192         USART1_string_send("send sms command error!\n");
193     }
194     else if(b_ok==1)
195     {
196         USART1_string_send("send sms OK!\n");
197     }
198     else if(b_ok==2)
199     {
200         USART1_string_send("send sms time out!\n");
201     }
202 }

447 unsigned char send_txt_sms(char *numb, char *msg)
448 {
449     unsigned char check_ok=0,i=0;
450     char *equal;
451
452     clae_r_u2rx_all();
453     USART2_string_send("AT+CMGF=1\r"); //设置发送格式是文本
454     GSM_delay_1ms(400);
455     USART2_string_send("AT+CSCS="); //设置TE字符7位默认字符,短信编码7位
456     USART2_send("");
457     USART2_string_send("GSM");
458     USART2_send("");
459     USART2_send('\r');
460     GSM_delay_1ms(400);
461     USART2_string_send("AT+CMGS="); //发送目标号码命令
462     USART2_send("");
463     USART2_string_send(numb); //目标号码 10086
464     USART2_send("");
465     USART2_send('\r');
466     GSM_delay_1ms(400);
467     equal=strstr(RxBuf_2,">"); //判断是否是发送标志
468     if(equal)
469     {
470         USART2_string_send(msg); //短信内容
471         GSM_delay_1ms(200);
472         clae_r_u2_buf();
473         USART2_send(0X1A); //结束标志
474         GSM_delay_1ms(100);

```

```
475     for(i=0;i<80;i++) //5s
476     {
477         equal=strstr(RxBuf_2,"CMGS:");
478         if(equal)
479         {
480             check_ok=1;
481             break;
482         }
483         else
484         {
485             GSM_delay_1ms(100);
486             check_ok=2;
487         }
488     }
489 }
490 }
491 else
492 {
493     check_ok=0;
494 }
495 claer_u2rx_all();
496 return check_ok;
497 }
```

按键 2 是发送中文短信，号码和内容已经转为 UNICODE 编码，而且是字符格式的。下图是《SIM900A_AT 命令手册》里中文短信例子。格式有小小不同。注意的是，并不是把发送格式设为 PDU，而是文本格式即跟英文短信一样，不然会出错。

发送含汉字的短信。	AT+CSMP=17,167,2,25	OK
	AT+CSCS="UCS2 "	OK
	AT+CMGS="0031003300390031003800310038003x003x003x003x">4E014E50<Ctrl+Z>	+CMGS:36
		OK

UNICODE 编码是一种国际标准编码，由 2 个字节表示一个符号或文字。下图中，把 10086 这个号码转成 UNICODE 编码后长度增加一倍，转成字符后长度变成 4 倍。看代码，第一条就是设置短信格式还是文本格式，然后是设置参数，接着是字符编码。发送号码后会返回 ‘>’ 符号，程序没有检测就直接发内容，然后发结束符。这个过程很简单，一般不会出错，为了简化过程，对模块返回的信息不作检测也是可以的。

```

203  /******
204  发送中文短信。号码和短信内容以转换成UNICODE。
205  *****/
206  char Target_number2[]="00310030003000380036"; //目标号码10086
207
208  // 你好，这是短信测试例子' 短信内容的UNICODE编码
209  char Target_msg2[]="4F60597DFF0C8FD9662F77ED4FE16D4B8BD54F8B5B503002";
210
211  void send_sms_pdu_1(void)
212  {
213      unsigned char b_ok=0;
214      b_ok=send_pdu_sms(Target_number2,Target_msg2); //发中文短信
215      if(b_ok==1)
216      {
217          USART1_string_send("send pdu sms OK!\n");
218      }
219      else if(b_ok==2)
220      {
221          USART1_string_send("send pdu sms time out!\n");
222      }
223  }

```

```

624  /******
625  * 发送中文短信
626  *****/
627  char AT_CSCS2[]={ 'A','T','+','C','S','C','S','=',' ','U','C','S','2',' ','\r'};
628  unsigned char send_pdu_sms(char *numb,char *msg)
629  {
630      unsigned char check_ok=0,i;
631      char *equal;
632
633      clae_r_u2rx_all();
634      USART2_string_send("AT+CMGF=1\r"); //设置发送格式是文本
635      GSM_delay_1ms(400);
636      USART2_string_send("AT+CSMP=17,0,2,25\r");
637      GSM_delay_1ms(400);
638      USART2_string_send(AT_CSCS2); //设置TE字符为UNICODE
639      GSM_delay_1ms(400);
640      clae_r_u2_buf();
641      USART2_string_send("AT+CMGS="); //发送目标号码命令
642      USART2_send(" ");
643      USART2_string_send(numb); //目标号码10086
644      USART2_send(" ");
645      USART2_send("\r");
646      GSM_delay_1ms(400);
647      USART2_string_send(msg); //短信内容“你好”对应的UNICODE码，
648      GSM_delay_1ms(300);
649      clae_r_u2_buf();
650      USART2_send(0x1A); //结束标志
651      GSM_delay_1ms(100);

```

下图是设置中文短信发送的过程。发中文最大的问题是把短信内容和号码转成 UNICODE 格式，UNICODE 是有 2 个字节构成，如数字 1 的 ASCII 码是 0X31，转成 UNICODE 是 0X0031，由于是字符形式发送，即变成 4 个字节，‘0’ ‘0’ ‘3’ ‘1’；ASCII 的 UNICODE 是不变的，可直接转换，只需要高字节补 0 即可；但中文的国内编码是 GBK（GB2312）跟 UNICODE 编码是不相同的，也没有任何规律，只能通过查表获得。查表采用折半查找。

首先来看号码，调用 ‘number_to_unicode 函数’ 来转换，上面都讲过了，数字转换很简单补 0 即可。再把数字对应的 16 进制编码转成字符。

汉字字符串在 KEIL 里是 GBK 编码的，由于 UNICODE 里的汉字编码没有规律，只能通过查表来转换。这个编码表有 GBK 编码和对应的 UNICODE 编码组成。在表里 GBK 编码是有序排列的，由小到大排列。为了提高效率采用折半查表法。折半查表法的原理可以去百度查，这里不详细叙述。要注意一点是，无论是号码或者内容的转换，转换后的数据比原数据占的空间要大，因此定义数据缓冲区是要预留足够的空间。经常有人问，开发板为什么不用 89C51，而是用 STC15 大容量系列或 STM32 等大容量单片机。短信发送对单片机资源要求还是不能太低的。没有几百字节的 RAM 是不行的。

```

224 1/*****
225 发送中文短信。
226 *****/
227 char Target_number3[]="13812345678";
228 char Target_msg3[]="你好，这是短信测试例子，请删除。";
229
230 void send_sms_pdu(void)
231 {
232     unsigned char b_ok=0;
233     char number[50];           //存放UNICODE编码的手机号码
234     char sms[300];            //存放GBK转成UNICODE的编码字符串
235
236     number_to_unicode(Target_number3,number); //把号码转成UNICODE
237     gbk2312_to_unicode(Target_msg3,sms);      //短信内容由GBK转成UNICODE编码。
238     //USART1_string_send(sms);
239     b_ok=send_pdu_sms(number,sms); //发中文短信
240     if(b_ok==1)
241     {
242         USART1_string_send("send chinese sms OK!\n");
243     }
244     else if(b_ok==2)
245     {
246         USART1_string_send("send chinese sms time out!\n");
247     }
248 }

```

从 SIM 中读取短信。

GSM 接收到短信后默认是存在 SIM 里的。SIM 卡一般能储存 50 条短信，如果卡已经储存了 50 条短信，新的短信就不能被储存，等于收不到新短信了。因此收到新短信读取后就及时删除，以便接收新短信。从 SIM 卡中读取短信是根据短信的储存序号来读取。读短信一般以 PDU 格式读取，因为无法区别短信是中文还是英文内容。读取后根据格式解码，分离出号码、日期时间和短信内容。然后根据短信内容的编码方式解码。英文短信内容的编码方式是 7 位编码，中文是 UNICODE 编码。短信的开始是‘089’是标准的短信，常用于用户与用户之间的短信，‘03A1’一般代理商发的或彩信格式的短信。具体的短信编码解码要请参考《[短消息编码 PDU 格式解析](#)》。短信解码才是 GSM 中最复杂的。

当模块收到一条新短信时，默认是存在 SIM 里，而且上报短信位置。下图是收到新短信的提示和读取该短信。这个一条英文短信，设置格式为文本，然后发读取命令，就能清楚看到短信的内容、号码和日期。如果是中文短信就不能用这种方法。

自动上报收到短信通知 阅读此短信息。 说明：短信序号与收到的+CMTI 提示信息中的序号一致。	AT+CMGR=1	+CMTI: "SM",1 +CMGR: "REC UNREAD", "+8613918186089",,"02 /01/30,20:40:31+00"
----------------------------------------------------------	-----------	---------------------------------------------------------------------------------------

SMS 是由 ETSI 组织制定的一个规范。短信息格式有两种：TEXT 模式和 PDU 模式。TEXT 模式是基于 ASCII 码形式字符的一种结构模式，每一条命令很容易读懂，实现起来电十分容易；缺点是不能收发中文短信，PDU 模式也是基于十六进制形式字符的，数据和代码都经过编码，所以无法直接读懂；但 PDU 模式同时支持中英文两种短信。PDU 模式收发短信包括 3 种编码：7 位、8 位和 UCS2 编码。7 位编码用于发送普通的 ASCII 字符，8 位编码用于发送数据信息，UCS2 编码用于发送 Unicode 字符。PDU 模式在 GSM 移动设备中使用得最为普遍。

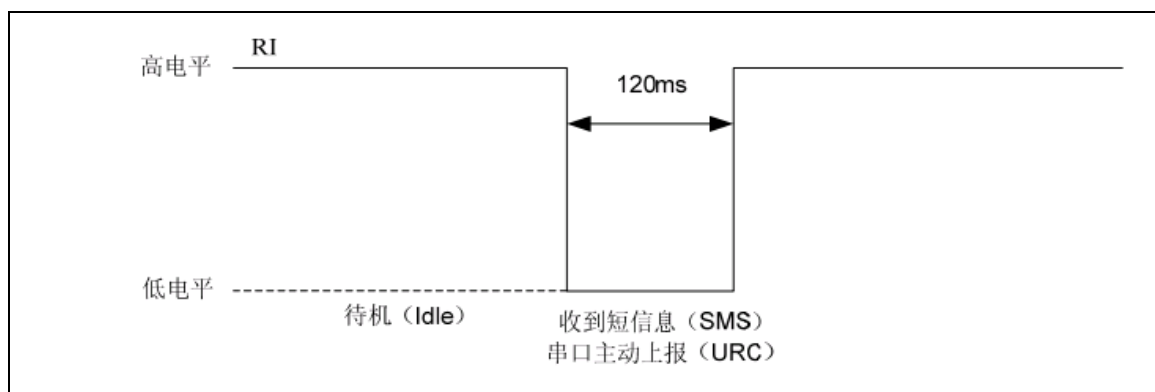
7 位编码算法。ASCII 字符的最大值是 127 即 16 进制的 0X7F。换成 2 进制最高位即第 8 位是 0；为了提高数据的有效传输，那把所有字符的最高位由一下个字符的最低位替代，组成新的 8 位数，8 个这正常的字符编码后变成 7 个字符。编码过程，将第 1 个字符转换为 7 位的二进制，将第 1 个字符右移 0 位。然后，将第 2 个字符的最右边的第 0 位加到第 1 个字符的第 7 位，形成一个 8 位字符，第 2 个字符的最右边的第 1 位通过右移 1 位方式销毁。之后，将第 2 个字符转换为 7 位的二进制，

将第 2 个字符右移 1 位；将第 3 个字符最右边的 2 位 (第 1、0 位) 填加到 第 2 个字符的第 7、6 位，形成一个 8 位字符；第 3 个字符的最右边的 2 位通过右移 2 位方式销毁。其他字符依此类推。

7 位编码解码。解码过程是将第 1 个字符的第 7 位移走，保存起来，补给第 2 字符的第 0 位。第 1 个字符左移 0 位，第 1 个字符的第 7 位销毁，形成一个新的 7 位 ASCII 字符；第 2 个字符第 7、6 位移走，保存起来，提供给第 3 个字符的第 2、1 位，第 2 个字符的第 7、6 位销毁，第 2 个字符左移 1 位，将前一个字符回填内容添加到第 2 个字符的第 0 位，从而又形成一个新的 7 位 ASCII 字符，依此方法类推。须注意的是，当第 8 个字符提供第 9 个字符全部 7 位时，形成第 9 个字符，同时第 7 位 提供给第 10 个字符的最右边第 0 位。依次循环，形成解码数据。

PDU 模式 UCS2 编码。从发短信的例程可知，是把 GBK 转成 UNICODE 编码的。译码过程也就是把 UNICODE 转成 GBK 编码。发短信时编码是查表获得，编码表是以 GBK 排序的，UNICODE 编码是无序的。现在译码，如果还是用那个表就不能用折半查找，只能老老实实从头找到尾，一个一个去对比。这样效率很低。或者再以 UNICODE 编码排序做一个表，这样就变成有 2 个表，查找效率会很高，但对单片机资源要求就很高，就是要有很大的代码存储空间 (FLASH)。

短信的接收可以通过**硬件判断**，也可以软件判断。硬件判断是根据 GSM 模块的 RI 引脚的电平来判断。当收到新短信，RI 引脚有一个 120 毫秒的低电平脉冲，注意有电话打进也有低电平，不同的是，短信是一个 120MS 的低电平脉冲，而电话是一直是低电平直到接通电话或挂断电话才恢复高电平。软件判断新短信是根据模块串口输出的信息是否包含 '+CMTI:' 这个标识来。硬件只能知道有新短信，但不知道储存位置，所以最终还要通过软件来确定位置，但就不需要用轮询的方式检测有没有新短信。开发板中的 RI 信号，平时是高电平，判断变成低电平脉冲的时间。



自动上报收到短信通知

+CMTI: "SM ",2

这里使用外部中断来实现。在程序初始化时设置好外部中断，有新短信时，触发中断，通过检测 RI 的低电平时间来识别是否短信，如果是短信，就检测串口收到的数据里是否含有新短信接收的字段标识，从中取得新短信的存储位置。

```

245  /******
246  void EXTI0_Config(void)
247  {
248      EXTI_InitTypeDef  EXTI_InitStructure;
249      GPIO_InitTypeDef  GPIO_InitStructure;
250      NVIC_InitTypeDef  NVIC_InitStructure;
251
252
253      /* Configure PB.13 pin as input floating */
254      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
255      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU; //上拉输入
256      GPIO_Init(GPIOB, &GPIO_InitStructure);
257
258      /* Enable AFIO clock */
259      //RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
260
261      /* Connect EXTI15-10 Line to PB.13 pin */
262      GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource13);
263
264      NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置先占优先级0位，响应优先级4位；
265      /* Configure EXTI0 line */
266      EXTI_InitStructure.EXTI_Line = EXTI_Line13;
267      EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
268      EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //下降沿
269      EXTI_InitStructure.EXTI_LineCmd = ENABLE;
270      EXTI_Init(&EXTI_InitStructure);
271

```

```

274  /******
275  *检查是否有电话打进，低电平表示有电话/短信，120ms低电平是短信
276  ******
277  unsigned char sim900_ring()
278  {
279      unsigned char ri=0;
280      if(GSM_RI()==1)
281          ri=0;
282      else
283      {
284          GSM_delay_1ms(150); //>120ms
285          if(GSM_RI()==1)
286              ri=2; //短信
287          else
288              ri=1; //电话
289      }
290      return ri;
291

```

```

326      else if (ring==2) //短信
327      {
328          SIM900_ring=2;
329          i=1;
330          GSM_delay_1ms(10);
331          same=strstr(RxBuf_2, "CMTI:"); //检查短信  sms_new[20];
332          if(same)
333          {
334              SIM900_sms[0] +=1;
335              *same = 'A';
336              same +=11;
337              if ((*same>='0') && (*same<='9'))
338              {
339                  i= SIM900_sms[0];
340                  SIM900_sms[i] =(*same-0x30);
341                  *same++;
342                  if ((*same>='0') && (*same<='9'))
343                  {
344                      SIM900_sms[i] *=0x10;
345                      SIM900_sms[i] +=(*same-0x30);
346                  }
347              }
348          }
349      }
350

```

4.连接 GPRS，TCP/IP 应用。参考《4-GPRS 连接-stm32》例程。

当 GPRS、TCP/IP 应用很多人问这个网络协议是怎样工作的，有没有教程？网络分几层？有没有开发包。为什么会出现这个现象，是因为大家都把这个 GPRS 当做电脑用的以太网了。手机无线网跟 WIFI 或有线以太网是不一样的，首先一点是手机网有几个频段，如 2 频和 4 频，第二个是制式，有 GSM，CDMA 等。这些都是由手机厂家和运营商的设备供应商制定的，而且不会公开。厂家把这个网络通信协议集成到 GSM 模块里，用户只需要按规定发指令来连接使用。

在默认下模块是处于非透传模式的。先用 AT+CGATT?来确认是否付着 GPRS 网络。然后发送连接命令，SIM900A 模块是支持 IP 地址或域名直接连接的。如命令 AT+CIPSTART="TCP","www.baidu.com",80; 命令包括连接协议 TCP 或 UDP，然后是域名或 IP 地址，接着是连接端口；连接成功后返回 CONNECT OK，失败返回 CONNECT FAIL。连接成功后输入发送命令才能发数据，数据发送跟短信有点相同，都是以 'CTRL+Z' 结束。要退出连接发 'AT+CIPSHUT 回车'。要注意的是，如果连接目标不存在或线路问题，模块会等待很久约 100 秒才提示连接失败，而且这个时间是不可设置的。开发设计时可以不等这么久，设置一个较少的时间，如 20 秒，如果过 20 秒了还不能连接上，就发退出连接命令停止连接。

```
AT+CIPSTART="TCP","echo.u-blox.com",7 //建立连接
OK

CONNECT OK
```

```
249  /******
250  *创建链接,
251  *****/
252  char AT_IP[]="www.baidu.com";
253  char AT_PORT[]="8001";
254  //-----
255  void at_create_connect(void)
256  {
257
258      unsigned char status=0;
259
260      status = sim900_at_cgatt();
261      if(status==1) //支持GPRS
262      {
263
264          status=0;
265          SIM900.ip_mod=1; //非透传模式
266          sim900_at_cipmode(1); //
267          status=tcipip_connect(AT_IP,AT_PORT,1); //
268          if((status==0)|| (status==2)) //规定时间内没有链接成功
269          {
270              AT_CIPSHUT(); //退出连接
271              USART1_string_send("Links time out!\n");
272              SIM900.ip_connect=0;
273          }
274          else if(status==1)
275          {
276              USART1_string_send("Links Success!");
277              SIM900.ip_connect=1;
278          }
279      }
280      else
281      {
282          USART1_string_send("Does not support GPRS!");
283      }
284  }
```

```

438  /*****
439  *非透传发送数据
440  *****/
441
442  unsigned char AT_CIPSEND(char *dat)
443  {
444      char *equal;
445      unsigned char count,status=0;
446
447      USART2_string_send("AT+CIPSEND\r");    //发送数据
448      GSM_delay_1ms(300);
449      USART2_string_send(dat);    //不要以回车结束
450      USART2_send(0x1A);    //发CTRL+Z结束

```

5.透明传输，连接 GPRS，TCPIP 应用。参考《5-GPRS 透传链接-stm32》例程。

什么是透明传输？就是不管你发什么数据给 GSM 模块，模块就把这些数据完全发送出去，没有返回值；举个例子：如 QQ 之间通信，输入文字信息后点发送，对方就会收到你发的信息，但你不知道信息是如何经过网络传输的。直接点就是，只管输入数据，不用理会后台是如何操作的。连接过程跟非透传是一样的。

创建连接前，先设置为透传模式，即 AT+CIPMODE=1，接着是连接，连接成功返回是‘CONNECT’没有带‘OK’，这是软件判断；硬件判断是模块的‘DCD’引脚。DCD 有 2 中状态，命令状态高电平和数据状态低电平。命令状态是指输入 AT 指令有数据返回，数据状态是指模块处于透传模式下连上了 GPRS，无论发什么数据包括 AT 指令在内的，都不会有返回数据，数据被发送出去。根据 DCD 引脚的电平检查 GSM 模块所处的状态，使用硬件检测状态可以让 CPU 省时省资源，而且很可靠。如果要退出，但处于透传下不能使用 AT 指令，怎么办？可以通过发 3 个加号‘+++’不带回车，即可切换到命令模式，再发退出指令‘AT+CIPSHUT’退出连接。在连接的过程中其中一边断开，模块返回‘CONNECT CLOSE’，DCD 引脚电平也变高。

设置命令	响应
AT+CIPMODE=	OK
<mode>	ERROR
参数说明	
<mode>	0 非透明模式
	1 透明模式

DCD：主要用于 PPP 拨号、透传功能下，判断模块处于数据态还是命令态；
命令态——2.8V 电平；
数据态——低电平

2.2.12 +++ 从数据模式或 PPP在线模式切换至命令模式

+++ 从数据模式或 PPP 在线模式切换至命令模式

执行命令

+++

响应

+++ 字符序列可使 TA 忽略当前 AT 接口的数据传输，并切换至命令模式。它允许 TA 在保持与远端服务器数据连接的状态下，仍然可输入 AT 命令。

OK

为避免 +++ 被错误的识别为数据，需要遵循以下步骤：

- 1.“+++”输入前 T1 时间（1 秒）内无字符输入。
- 2.连续输入“+++”，中间不能有其他字符，并且输入+号之间不能超过 0.5 秒。
- 3.“+++”输入后 T1 时间（0.5 秒）内无字符输入。
- 4.切换至命令模式，否则重新进入步骤 1。

6.GSM 应用-温度监控短信报警。参考《6-温度短信报警-stm32》例程。

温度采集采用 DS18B20 这个元件。通信方式采用单总线，采集温度范围-55~+125 摄氏度。元件的详细参数请看技术文档，这里不作介绍。这里只对数据的读取做说明。下图是 DS18B20 的复位和数据读写时序。

DS18B20 的复位时序如下：

- 1.单片机拉低总线 480us~950us，然后释放总线（拉高电平）。
- 2.这时 DS18B20 会拉低信号，大约 60~240us 表示应答。
- 3.DS18B20 拉低电平的 60~240us 之间，单片机读取总线的电平，如果是低电平，那么表示复位成功。
- 4.DS18B20 拉低电平 60~240us 之后，会释放总线。

DS18B20 写逻辑 0 的步骤如下：

- 1.单片机拉低电平大约 10~15us。
- 2.单片机持续拉低电平大约 20~45us 的时间。
- 3.释放总线

DS18B20 写逻辑 1 的步骤如下：

- 1.单片机拉低电平大约 10~15us。
- 2.单片机拉高电平大约 20~45us 的时间。
- 3.释放总线

DS18B20 读逻辑 0 的步骤如下：

- 1.在读取的时候单片机拉低电平大约 1us
- 2.单片机释放总线，然后读取总线电平。
- 3.这时候 DS18B20 会拉低电平。
- 4.读取电平过后，延迟大约 40~45 微妙

DS18B20 读逻辑 1 的步骤如下：

- 1.在读取的时候单片机拉低电平大约 1us
- 2.单片机释放总线，然后读取总线电平。
- 3.这时候 DS18B20 会拉高电平。
- 4.读取电平过后，延迟大约 40~45 微妙

首先设置单片机 IO 为开漏输出，读数据为什么还要设成输出？这是因为单总线的缘故，即同一条数据线上即发送数据又接收数据，单片机设成输入就不能输出数据，但设成输出也能读取数据。设成开漏意味单片机 IO 没有任何驱动能力，就不会增加信号线上的负载能力，减少电平变化的影响，就不会影响读取数据。

现在来看 DS18B20 读取数据个过程：

- ① 先复位信号线。如果 DS18B20 正常工作，就会响应拉低数据线电平。
- ② 然后开发发命令，发 0xCC，跳过 ROM 指令。这是因为信号线上只有一个元件，不需要去

对比匹配序列号。

- ③ 接着发读取数据命令，发 0xBE 或者 0xEE，获取数据，2 个命令有什么区别？0xBE 命令是读取 9 个缓冲区数据，里面包括温度数据和校验数据。0xEE 命令只读温度数据的 2 个缓冲区。
- ④ 读取数据，如果不校验，数据只读 2 个就可以了。
- ⑤ 重新复位数据线。
- ⑥ 发跳过 ROM 指令 0xCC；
- ⑦ 然后发温度转换开始命令 0x44；

```

102 /*****
103 读取温度 (DS18B20)
104 *****/
105 unsigned char Read_Temperature(unsigned char ch)
106 {
107     unsigned char temp_err=0;
108     unsigned int t;
109     if(1)
110     {
111         temp_err = ow_reset(ch);
112         if(temp_err==1) //发生错误
113             temp_err = ow_reset(ch);
114         write_byte(0xCC, ch); // Skip ROM
115         write_byte(0xBE, ch); // Read Scratch Pad
116         for(t=0;t<9;t++)
117         {
118             ds[t]=read_byte(ch); //接收低位
119         }
120
121         ow_reset(ch); //
122         write_byte(0xCC, ch); //Skip ROM
123         write_byte(0x44, ch); // Start Conversion
124     }
125
126     return temp_err;
127 }
128
    
```

从操作过程来看很简单。再细心看就有一小问题，就是读完数据才开始转换。这是什么原因呢？这是因为为了提高单片机的执行代码的效率，和简化程序，缺点是第一次读取的数据不正确而被丢弃。还有一点要注意的，就是 DS18B20 每次转换温度数据要 600 毫秒，就是说单片机每次读取温度数据的时间间隔不能少于 600 毫秒。

从 DS18B20 读取到数据后，要转换成真实的温度数据，正温度数据转换比较直接。负温度就要多一步运算。计算后得到的数据，用最高位来作区分正负，最高位为 1 是负温度。

温度/数据关系 表 2

温度 ℃	数据输出（二进制）	数据输出（十六进制）
+125	0000 0111 1101 0000	07D0h
+85	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Eh
-55	1111 1100 1001 0000	FC90h

```

168  /*****
169  *读取温度 and 数据处理
170  *****/
171  unsigned short get_ds18b20()
172  {
173      unsigned char crc_ds;
174      unsigned short ds_temp=0;
175      Read_Temperature(1);
176      crc_ds=CRC_Data_ds18b20(ds); //CRC校验
177      if(crc_ds==0)
178      {
179          ds_temp=ds[1];
180          ds_temp <<=8;
181          ds_temp +=ds[0];
182          if(ds_temp>0xfc80) //温度零度以下
183          {
184              ds_temp=0xffff-ds_temp+1; //取数值
185              ds_temp= (ds_temp*5)/8; //转为实际温度值
186              ds_temp |= 0x8000;
187          }
188          else if(ds_temp<0x07d1) //温度0度以上
189          {
190              ds_temp= (ds_temp*5)/8; //转为实际温度值,比实际大10倍
191          }
192          else
193              ds_temp =0xffff;
194      }
195      else
196          ds_temp =0xffff;
197      return ds_temp;
198  }

```

下面的图是温度报警的代码，首先定义要发送的目标号码，和需要提示的文字。温度上限设定为 38 度。

第一步是读取温度，然后与设定值对比，超过就发短信报警。

第二步，把温度数据插到已定义字符串里，组成一条短信内容。然后发送。

第三步判断短信的发送结果。

这里完成了一个温度过高的报警过程，实际应用中有注意的地方。温度过高很可能延续很长时间，如果不断的短信报警，会花费很大，而且严重扰乱短信接收者的生活，那怕用电脑接收也会造成很大数据量。因此短信报警必需隔一段时间才能发一次。在程序上作一个延时，每发一次短信就产生一个延时。

```

226  /*****
227  温度过高短信报警
228  *****/
229  char Targ_number[]="13812345678";
230  char Targ_msg[]="现在温度是: ";
231  char Targ_msg2[]="度,温度过高报警.";
232  static unsigned short t_warn=380; //实际是38.0度
233  void temperature_alarm(void)
234  {
235      static unsigned char t_time=0;
236      unsigned short dt=0;
237      unsigned char send_s=0;
238      char t_string[5];
239      char t_msg[100];
240      dt=get_ds18b20(); //读取温度
241      if(dt>t_warn)
242      {
243          strcpy(t_msg,Targ_msg); //字符串复制
244          sprintf(t_string,"%d",dt); //把16进制数字转成10进制的字符串
245          strcat(t_msg, t_string); //字符串合并
246          strcat(t_msg, Targ_msg2); //字符串合并
247          USART1_string_send(t_msg); //
248          if(t_time==0) //为了避免不断发短信,每次发短信后等5分钟才能发下一次。
249          {
250              send_s=send_sms_chinese(Targ_number,t_msg); //发送短信
251              if(send_s==1)
252              {
253                  USART1_string_send("\nsend sms OK!\n");
254              }
255              else if(send_s==2)
256              {
257                  USART1_string_send("\nsend sms time out!\n");
258              }
259              t_time=150; //短信间隔计时,约5分钟。
260          }
261      }
262      if(t_time>0)
263      {
264          t_time--;
265      }
266  }

```

7.模拟量采集（ADC）。参考《7-GPRS 模拟量采集-stm32》例程。

这里是利用 STM32 单片机自身所带的 ADC 转换功能采集电压和电流信号。单片机自带的 ADC 是 12 位的，数据范围是 0-4095，参考电压是 3.3V，那么分辨率就是 $3300/4096=0.80566\text{mV}$ 。开发板上的电压信号输入范围是 0-10V，有 2 路电压源的输入端口。由于输入电压远高于参考电压，因此通过电阻分压后才读取数据。电流输入范围是 0-20mA，也有 2 路端口，用 120 欧姆电阻来取样。

首先是初始化 ADC，打开 ADC 时钟，配置对应的引脚为模拟输入，数据格式为右对齐，最后是设置转换时间。使用 ADC 前先校正，完成后就可以转换。电压和电流一起共有 4 个通道，采用逐个切换的方式读取 ADC 值。

读取到 ADC 值后，就要转换成实际的电压值。由于电压信号是经过分压所得，因此还要再一步转换。而电流信号是直接获取的，但单位不同还是要作一次转换。

如果 GSM 模块已经处于 GPRS 连接状态就直接发送数据，否则就先创建连接。

```

149 1/*****
150 3300/4096=0.80566=0.8, 读取ADC值并转换成真实电压值
151 *****/
152 void read_v_a(void)
153 {
154     unsigned int val=0;
155     static unsigned char ch=1;
156
157     val=get_adc_val();
158     val=val*806/1000;    //真实电压值
159     if(ch<3)
160         val=val*4;
161     adc_val[ch-1]=(unsigned short)val;
162     ch++;
163     if(ch>4)
164         ch=1;
165     adc_channel_sel(ch);
166     GSM_delay_1ms(50);
167
168     /* Start ADC1 Software Conversion */
169     ADC_SoftwareStartConvCmd(ADC1, ENABLE);
170
171 }

```

```

189 1/*****
190 通过GPRS发送出去
191 *****/
192 char unit_v[]={"mV,"};
193 char unit_a[]={"mA,"};
194
195 void adc_send_gprs(void)
196 {
197     char t_s[6];
198     char ADC_TP[50];
199
200     sprintf(t_s,"%d",adc_val[0]);    //把数值转成字符
201     strcpy(ADC_TP, t_s);             //合并成字符串。
202     strcat(ADC_TP, unit_v);
203     sprintf(t_s,"%d",adc_val[1]);
204     strcat(ADC_TP, t_s);
205     strcat(ADC_TP, unit_v);
206     sprintf(t_s,"%d",adc_val[2]/120);
207     strcat(ADC_TP, t_s);
208     strcat(ADC_TP, unit_a);
209     sprintf(t_s,"%d",adc_val[3]/120);
210     strcat(ADC_TP, t_s);
211     strcat(ADC_TP, unit_a);
212     USART1_string_send("ADC collect:");
213     USART1_string_send(ADC_TP);
214     USART1_send('\n');
215
216     if(SIM900.ip_connect==2)    //GPRS以连接
217     {
218         USART2_string_send("ADC collect:");
219         USART2_string_send(ADC_TP);    //发送数据
220         USART2_send('\n');
221     }
222 }

```

8.利用短信来控制继电器。参考《8 短信控制继电器-stm32》例程。

远程控制开关在实际应用中比较常见。一般距离不是很远，如几十米或几百米的就用一般无线来控制。但有时候距离离得很远几公路甚至跨省跨市的，就不好解决了。GSM 网络是一个基本覆盖全国所有地方的网络。用来控制设备那就容易很多。用短息控制开关是其中一种应用方式。优点是不受距离限制，网络可靠，操作简单；缺点是实时性不高。

有新短信到来会触发中断，中断函数记录位置后，由主循环程序来读取和解码。在

sim900_ring_check()函数里调用 read_sms_dtu()解码短信。短信解码后, check_sms_cmd()会对短信进行分析, 对比号码和短信内容是否跟设定参数一样, 如果号码匹配, 获取命令并进行对应的操作。

```

138  /*****
139  判断短信内容，获取命令
140  *****/
141  char sell_numb[]="13812345678";    //授权号码
142  char msg_cmd[]="继电器：断开。";   //协议
143  char msg_cmd2[]="继电器：闭合。";  //协议
144  void check_sms_cmd(void)
145  {
146
147      char *cd=0;
148
149      cd=strstr(phone.number,sell_numb);    //是否指定的号码
150      if(cd)
151      {
152          cd=strstr(phone.message,msg_cmd); //对比内容
153          if(cd)
154          {
155              LAY_OFF();    //关闭LED
156              LED_2_OFF();  //关闭继电器
157          }
158          else
159          {
160              cd=strstr(phone.message,msg_cmd2); //对比内容
161              if(cd)
162              {
163                  LAY_ON();    //打开继电器
164                  LED_2_ON();   //打开LED
165              }
166          }
167      }
168  }
169  }

```

9.RTC 读取。参考《8 短信控制继电器-stm32》例程。

RTC 采用 pcf8563 时钟芯片。数据接口是 IIC，具体的技术参数和时序请看技术文档。

第一步设置通信 IO 位开漏输出，PCF8563 init();

第二步启动时钟; startclk();

第三步读取时钟数据数据，read rtc time();。

10.GSM 定位 (AGPS)。参考《51 gsm 基站定位 (AGPS)》例程。

这个功能不是所有的 GSM 模块都有的，市场上 SIM900A 有 3 个版本，区别在与是否带彩信（MMS）、定位（LBS）和文本转语音（TTS）；普通版本的是这 3 个功能都不带，带彩信（MMS）和定位（LBS）是一个版本；带文本转语音（TTS）是又一个版本。发送命令顺序如下图。

3.1 Activate bearer profile

```
AT+SAPBR=3,1,"Contype","GPRS"           //Set bearer parameter
```

OK

```
AT+SAPBR=3,1,"APN","CMNET"
```

OK

```
AT+SAPBR =1,1                             // Activate bearer context
```

OK

```
AT+SAPBR=2,1
```

```
+SAPBR: 1,1,"10.89.193.1"
```

OK

3.2 Get location

```
AT+CIPGSMLOC=1,1
```

```
+CIPGSMLOC: 0,121.354848,31.221402,2011/01/26,02:41:06
```

OK

3.3 Deactivate bearer profile

```
AT+SAPBR=0,1                             // Deactivate bearer context
```

OK

11.其他功能请查看 AT 命令手册。或其他功能说明手册。

谢谢支持！

飞瀑浪潮电子淘宝店